**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационной безопасности**

отчет

**по лабораторной работе №1**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: «**Операции над конечными неупорядоченными множествами**»

|  |  |  |
| --- | --- | --- |
| Студенты гр. 3364 |  | Юсфи А. |
| Преподаватель |  | Абросимов И.К. |

Санкт-Петербург

2024

**ЗАДАНИЕ НА ЛАБОРАТОРНУЮ РАБОТУ**

|  |
| --- |
| Студенты: Юсфи А. |
| Группа: 3364 |
| Тема работы:  Операции над конечными неупорядоченными множествами. |
| Цель работы:  Экспериментально оценить время сложность алгоритма операций над конечными неупорядоченными множествами по его реализации на языке программирования C++. |
| Содержание пояснительной записки:  Теоретическое введение, решение поставленной задачи, заключение, приложение – исходный код программы. |

|  |  |  |
| --- | --- | --- |
| Студенты гр. 3364 |  | Юсфи А. |
| Преподаватель |  | Абросимов И.К. |

**ТЕОРЕТИЧЕСКОЕ ВВЕДЕНИЕ**

* 1. **Формулировка определения множества**

Множество представляет собой совокупность объектов, называемых элементами множества и удовлетворяющих заданным требованиям.

Говорят, что принадлежит , и пишут , в противном случае – .

Объединение, пересечение, разность множеств:

;

;

.

где

* 1. **Алгоритмы операций над конечными неупорядоченными множествами**
     1. **Алгоритм объединения конечных неупорядоченных множеств**

Алгоритм заключается в следующих шагах:

Шаг 1. Проверить, что оба множества имеют одинаковую структуру и элементы;

Шаг 2. Создать новое множество для хранения результата объединения;

Шаг 3.  Последовательно обработать каждый элемент бинарных представлений обоих множеств и объединить их с помощью логической операции «ИЛИ».

Шаг 4. Сохранить полученное объединение в результирующее множество

* + 1. **Алгоритм пересечения конечных неупорядоченных множеств**

Алгоритм заключается в следующих шагах:

Шаг 1.  Проверить, что оба множества имеют одинаковую структуру и элементы;

Шаг 2. Создать новое множество для хранения результата пересечения;

Шаг 3. Последовательно обработать каждый элемент бинарных представлений обоих множеств и найти общие элементы с помощью логической операции «И».

Шаг 4. Сохранить результат пересечения в результирующее множество

* + 1. **Алгоритм разности конечных неупорядоченных множеств**

Алгоритм заключается в следующих шагах:

Шаг 1. Проверить, что оба множества имеют одинаковую структуру и элементы;

Шаг 2. Создать новое множество для хранения результата разности;

Шаг 3.  Инвертировать элементы второго множества, которое вычитается из первого;

Шаг 4. Последовательно обработать каждый элемент бинарных представлений и найти разность с помощью логической операции «И».

Шаг 5. Сохранить результат разности в результирующее множество;

**1.3 Методика экспериментального определения временной сложности.**

Временная сложность – отображение, ставящее в соответствие размеру входа количество операций.

Формализм, описывающий множество функций, включающее в себя временную сложность – символ «о большое».

Временная сложность ищется с точностью до константного множителя. При этом ищется минимальная с точки зрения скорости роста верхняя оценка временной сложности.

Методика экспериментального определения временной сложности:

Шаг 1. Зафиксировать размер входных данных ;

Шаг 2. Рассчитать время каждого выполнения алгоритма ;

Шаг 3. Из отношения ,найти константу 𝑎, характеризующую скорость роста временной сложности. Если отношения даже при не являются примерно равными, то это означает, что функция сложности содержит факториал;

Шаг 4. Решить относительно 𝑓 систему уравнений, учитывая класс функций, к которому принадлежит временная функция:

**1.4 Описание структуры данных**

Поля структуры данных:

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Тип данных** | **Область видимости** | **Назначение** |
| \*BinnarryArray | Word\* | private | Указатель на массив типа word, представляющий основной контейнер для хранения данных. |
| NumberOfWords | int | private | |  | | --- | | Количество слов (word) в массиве BinnarryArray |  |  | | --- | |  | |
| NumberOfelemant | int | private | Количество элементов, добавленных в множество. |

|  |  |  |  |
| --- | --- | --- | --- |
| SizeDecimaleForSet | Int | Private | Размер множества. |
| \*SetDecimale | Int\* | Private | |  | | --- | |  |  |  | | --- | | Указатель на массив, представляющий множество в десятичном виде. | |
| SizeOfWord | static constexpr int | Private | Представляет количество бит в одном элементе типа `word`, используется для вычисления индексов и смещений при доступе и манипуляции битами в бинарном представлении множества. |

Методы структуры данных:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Название** | **Принимаемые аргументы** | **Возвра-щаемый тип данных** | **Область видимости** | **Назначение** |
| KaderStruct(); | - | - | public | Конструктор по умолчанию. |
| KaderStruct(); | int\* array, unsigned int size | - | public | Конструктор, инициализирующий структуру с заданным массивом и его размером |
| ~KaderStruct(); | - | - | public | Деструктор. |
| int getNumOfWords(); | - | - | public | Возвращает количество элементов, используемых в массиве binArr. |
| int GetNumbersOfelemant(); | - | - | Public | Возвращает количество элементов находящихся в множестве. |
| int GetTheSizeOfthedecimale(); | - | - | Public | Возвращает размер множества в десятичном представлении. |
| int getNumOfWords(); | - | int | Public | Возвращает количество элементов, используемых в массиве binArr. |
| bool setBit(); | int offset, bool newBit | bool | Public | Устанавливает бит в указанном смещении в заданное значение . |
| bool getBit(); | int offset | bool | Public | Возвращает значение бита в указанном смещении. |
| bool invertBit(); | int offset | bool | Public | Инвертирует бит в указанном смещении. |
| bool has(); | int num | bool | Public | Проверяет, существует ли заданное число в множестве. |
| bool add(); | - | bool | Public | Добавляет число в множество. |
| bool discard(); | int num | bool | Public | Удаляет число из множества. |
| int powerOfSet(); | - | int | Public | Возвращает кардинальность (количество элементов) множества. |
| void clear(); | - | void | Public | Очищает все элементы в множестве. |
| void clearByWord(); | int indElem | - | Public | Очищает конкретное слово в бинарном массиве. |
| void fillRandomSet(); | - | - | Public | Заполняет множество случайным набором чисел. |
| void showSet(); | - | - | public | Отображает элементы множества в десятичном представлении. |
| void showSetBin(); | - | - | public | Отображает бинарное представление множества. |
| int hasInDec(); | int num | int | Public | Находит индекс числа в десятичном массиве |
| word\* accessWordByIndex() const; | int index | - | Public | Предоставляет доступ к конкретному слову в бинарном массиве. |
| KaderStruct\* inversionSet() const; | - | - | Public | Возвращает дополнение (инверсию) множества. |
| KaderStruct\* unionSet() const; | const KaderStruct& other | - | Public | озвращает объединение текущего множества с другим множеством. |
| KaderStruct\* intersectionSet() const; | const KaderStruct& other | - | Public | Возвращает пересечение текущего множества с другим множеством. |
| KaderStruct\* subtractionSet() const; | const KaderStruct& other | - | Public | Возвращает разность (вычитание) текущего множества из другого множества. |

**РЕШЕНИЕ ПОСТАВЛЕННОЙ ЗАДАЧИ**

**2.1 Сведения о языках программирования и ПО**

Во время выполнения курсовой работы был использован язык программирования С++. Установленная операционная система Windows 11 Pro. Средой разработки, которая применялась, является Clion версии 2024.2.4 с компилятором С++.

**2.1 Сведения о реализованных функциях**

Функция main.

Функция main демонстрирует работу класса KaderStruct, предназначенного для выполнения операций над конечными неупорядоченными множествами.

Исходный код содержится в файле main.cpp.

Объявление функции:

void main()

Тип функции: int.

Функция возвращает 0 при успешном завершении .

**2.3 Сведения о работе программы**

После запуска программы в консоль выводятся два случайно заполненных начальных множества в бинарном и десятичном представлениях. Затем выводятся три результата операций над множествами — объединение, пересечение и разность — в их бинарном и десятичном форматах (Рисунок 1).

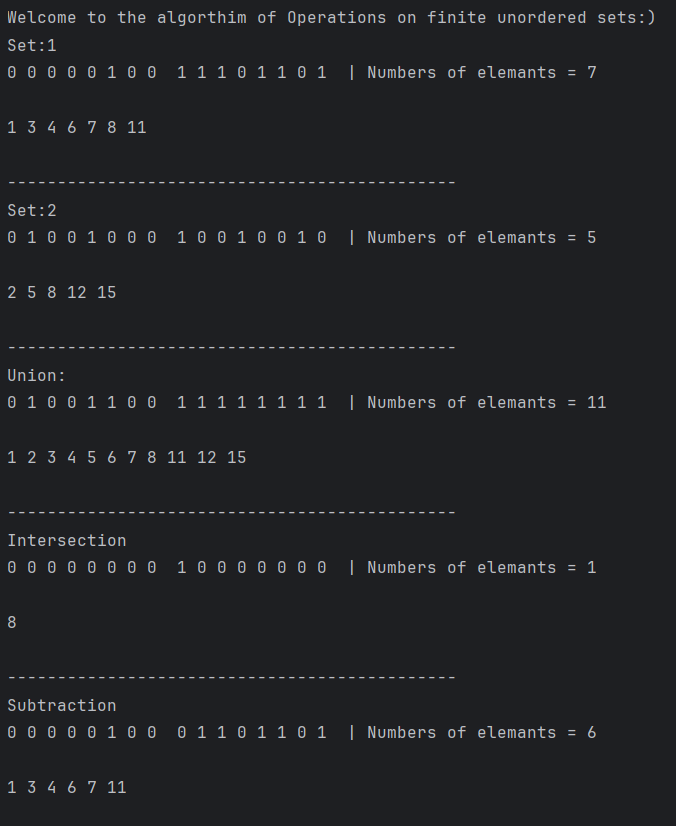


Рисунок 1 – тестирование программы

**2.4 Проведение расчетов по определению оценить время сложность алгоритма**

Для оценки временной сложности алгоритма следует выбрать наиболее затратную по времени операцию (разность) .

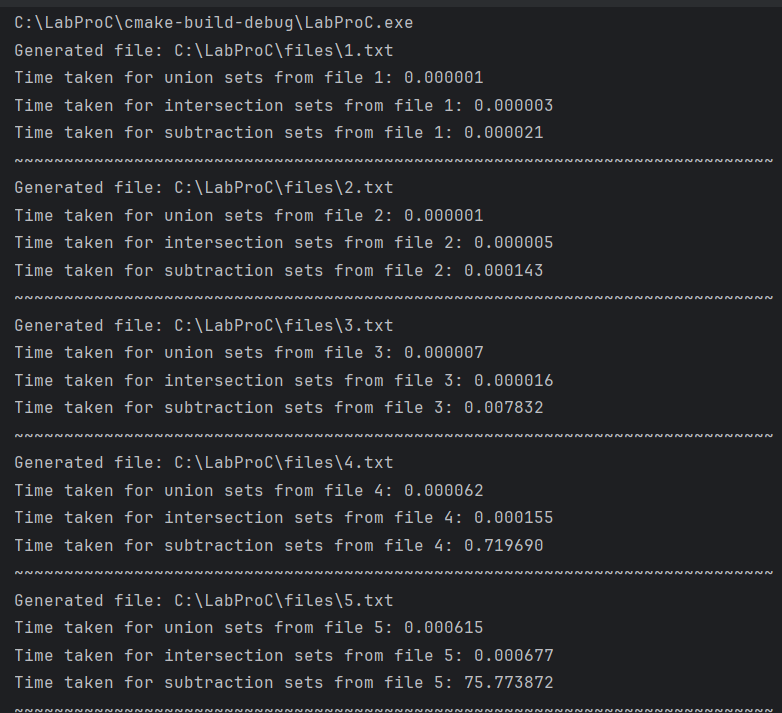


Рисунок 1 – Время выполнения операций

Таким образом, самая затратная по времени операция — операция разности множеств. Используя её, проведём оценку временной сложности.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 |
|  | 100 | 1000 | 10000 | 100000 | 1000000 |
| (мс) | 0.021 | 0.143 | 7.832 | 719.69 | 75,773.872 |
|  | 6.81 | 54.83 | 91.79 | 105.3 | - |

Таким образом, .

, откуда

Наша сложность – .

Построим эталонный и экспериментальный графики пространственной сложности (Рисунок 2).

Рисунок 2 – экспериментальный и эталонный графики пространственной сложности

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения лабораторной работы была экспериментально оценена временная сложность алгоритма операций над конечными неупорядоченными множествами, которая принадлежит классу линейных функций , где – количество элементов множества.

**ПРИЛОЖЕНИЕ**

**ИСХОДНЫЙ КОД ПРОГРАММЫ**

Файл заголовок структуры KaderStruct:

#ifndef KADERSTRUCT\_H

#define KADERSTRUCT\_H

typedef unsigned char word;

class KaderStruct {

private:

word \*binArr{};

int numOfWords;

int numOfElems;

int sizeDec;

int\* setDec{};

public:

static constexpr int sizeOfWord = sizeof(word) \* 8;

KaderStruct();

~KaderStruct();

KaderStruct(int\* array, unsigned int size);

int getNumOfWords();

int getNumOfElems();

int getSizeDec();

bool has(int num);

bool isEmpty();

bool add(int num);

int hasInDec(int num);

bool discard(int num);

int powerOfSet();

void clear();

void clearByWord(int indElem);

void fillRandomSet();

void showSetBin();

void showSet();

KaderStruct\* inversionSet() const;

KaderStruct\* unionSet(const KaderStruct& other) const;

KaderStruct\* intersectionSet(const KaderStruct& other) const;

KaderStruct\* subtractionSet(const KaderStruct& other) const;

word \*accessWordByIndex(int index) const;

bool setBit(int offset, bool newBit);

bool getBit(int offset);

bool invertBit(int offset);

};

#endif //KADERSTRUCT\_H

Файл структуры KaderStruct:

#include "KaderStruct.h"  
#include <stdexcept>  
#include <random>  
#include <chrono>  
#include <iostream>  
KaderStruct::KaderStruct(int\* array, unsigned int size) : sizeDec(size), setDec(array) {  
 if (array == nullptr || size <= 0) {  
 throw std::invalid\_argument("invalid arguments");  
 }  
 numOfElems = 0;  
 numOfWords = (size - 1) / *sizeOfWord* + 1;  
 binArr = new word[numOfWords]();  
}  
KaderStruct::~KaderStruct() {  
 delete[] binArr;  
}  
int KaderStruct::getNumOfWords() {  
 return numOfWords;  
}  
int KaderStruct::getNumOfElems() {  
 int totalElems = 0;  
 for (int i = 0; i < numOfWords \* *sizeOfWord*; i++) {  
 totalElems += (int) getBit(i);  
 }  
 numOfElems = totalElems;  
 return numOfElems;  
}  
int KaderStruct::getSizeDec() {  
 return sizeDec;  
}  
bool KaderStruct::getBit(int offset) {  
 if (offset < 0 || offset > numOfWords \* *sizeOfWord*) {  
 throw std::invalid\_argument("invalid arguments");  
 }  
 int elemInd = (offset) / *sizeOfWord*;  
 return binArr[elemInd] & (1 << (offset % *sizeOfWord*));  
}  
bool KaderStruct::setBit(int offset, bool newBit) {  
 if (offset >= numOfWords \* *sizeOfWord* || offset < 0) return false;  
 int elemInd = (offset) / *sizeOfWord*;  
 word \*elem = &binArr[elemInd];  
 if ((bool) (\*elem & (1 << (offset % *sizeOfWord*))) == newBit) {  
 return false;  
 }  
 \*elem ^= (1 << (offset % *sizeOfWord*));  
 numOfElems += (newBit) ? 1 : -1;  
 return true;  
}  
bool KaderStruct::invertBit(int offset) {  
 if (offset >= numOfWords \* *sizeOfWord* || offset < 0) return false;  
 int elemInd = offset / *sizeOfWord*;  
 word \*elem = &binArr[elemInd];  
 \*elem ^= (1 << (offset % *sizeOfWord*));  
 numOfElems += (1 << (offset % *sizeOfWord*)) ? 1 : -1;  
 return true;  
}  
bool KaderStruct::isEmpty() {  
 return getNumOfElems() == 0;  
}  
void KaderStruct::showSet() {  
 for (int i = 0; i < sizeDec; i++) {  
 if (getBit(i)) {

printf("%d ", setDec[i]);  
 }  
 }  
 std::cout << std::endl << std::endl;  
}  
void KaderStruct::showSetBin() {  
 for (int i = numOfWords \* *sizeOfWord* - 1; i >= 0; i--) {  
 printf("%d ", getBit(i));  
 if (i % *sizeOfWord* == 0) {  
 printf(" ");  
 }  
 }  
 printf("| Numbers of elemants = %d\n\n", getNumOfElems());  
}  
void KaderStruct::fillRandomSet() {  
 clear();  
 unsigned seed = std::chrono::system\_clock::*now*().time\_since\_epoch().count();  
 std::mt19937 generator(seed);  
 std::uniform\_int\_distribution<int> distribution(1, 10000);  
 for (int i = 0; i < sizeDec; i++) {  
 int random\_number = distribution(generator);  
 if (random\_number % 2 == 0) {  
 setBit(i, true);  
 }  
 }  
}  
void KaderStruct::clearByWord(int elemInd) {  
 if (elemInd < 0 || elemInd > numOfWords) return;  
  
 for (int i = 0; i < *sizeOfWord*; i++) {  
 numOfElems -= ((binArr[elemInd] & (1 << i)) ? 1 : 0);  
 }  
 binArr[elemInd] ^= binArr[elemInd];  
}  
void KaderStruct::clear() {  
 for (int i = 0; i < getNumOfWords(); i++) {  
 clearByWord(i);  
 }  
}  
bool KaderStruct::has(int num) {  
 int tempInd = 0;  
 for (int i = 0; i < sizeDec; i++) {  
 if (setDec[i] == num) {  
 tempInd = i;  
 break;  
 }  
 }  
 return getBit(tempInd);  
}  
bool KaderStruct::add(int num) {  
 int ind = hasInDec(num);  
 if (ind == -1) { return false; }  
  
 return setBit(ind, true);  
}  
bool KaderStruct::discard(int num) {  
 int ind = hasInDec(num);  
 if (ind == -1) { return false; }  
  
 return setBit(ind, false);  
}  
int KaderStruct::powerOfSet() {  
 return getNumOfElems();  
}  
  
int KaderStruct::hasInDec(int num) {  
 for (int i = 0; i < sizeDec; i++) {  
 if (setDec[i] == num) {  
 return i;  
 }  
 }  
 return -1;  
}  
word \*KaderStruct::accessWordByIndex(int index) const {  
 return &binArr[index];  
}  
KaderStruct \*KaderStruct::inversionSet() const {  
 KaderStruct \*resultSet = new KaderStruct(setDec, sizeDec);  
  
 for (int i = 0; i < numOfWords; i++) {  
 \*resultSet->accessWordByIndex(i) = ~(\*this->accessWordByIndex(i));  
 }  
 return resultSet;  
}  
KaderStruct\* KaderStruct::unionSet(const KaderStruct& other) const {  
 if (this->setDec != other.setDec) { return nullptr; }  
 KaderStruct \*resultSet = new KaderStruct(setDec, sizeDec);  
 for (int i = 0; i < numOfWords; i++) {  
 \*resultSet->accessWordByIndex(i) = \*other.accessWordByIndex(i)  
 | \*this->accessWordByIndex(i);  
 }  
 return resultSet;  
}  
KaderStruct \*KaderStruct::intersectionSet(const KaderStruct &other) const {  
 if (this->setDec != other.setDec) { return nullptr; }  
 KaderStruct \*resultSet = new KaderStruct(setDec, sizeDec);  
 for (int i = 0; i < numOfWords; i++) {  
 \*resultSet->accessWordByIndex(i) = \*other.accessWordByIndex(i) & \*this->accessWordByIndex(i);  
 }  
  
 return resultSet;  
}  
KaderStruct \*KaderStruct::subtractionSet(const KaderStruct &other) const {  
 if (this->setDec != other.setDec) { return nullptr; }  
 KaderStruct \*resultSet = new KaderStruct(setDec, sizeDec);  
 for (int i = 0; i < numOfWords; i++) {  
 \*resultSet->accessWordByIndex(i) = \*this->accessWordByIndex(i)  
 & \*other.inversionSet()->accessWordByIndex(i);  
 }  
  
 return resultSet;  
}  
  
KaderSt ruct::KaderStruct() : binArr(new word[0]), setDec(new int[0]), numOfWords(0), numOfElems(0), sizeDec(0) {  
  
}

Файл main:

#include <iostream>

#include "Structure/KaderStruct.h"

int main() {

int size = 16;

int \*arr = new int[size];

for (int i = 0; i < size; i++) {

arr[i] = i + 1;

}

KaderStruct ks = KaderStruct(arr, size);

KaderStruct ks1 = KaderStruct(arr, size);

printf("Welcome to the algorthim of Operations on finite unordered sets:) \n");

ks.fillRandomSet();//filling the first set

ks1.fillRandomSet();//filling the second set

printf("Set:1 \n");

ks.showSetBin();//print the first binary set

ks.showSet();//show the decimal first set

printf("---------------------------------------------\n");

printf("Set:2 \n");

ks1.showSetBin();// print the second binary set

ks1.showSet();//print the second set

printf("---------------------------------------------\n");

printf("Union: \n");

KaderStruct \*unionSet = ks.unionSet(ks1);//union

unionSet->showSetBin();

unionSet->showSet();

printf("---------------------------------------------\n");

printf("Intersection \n");

KaderStruct \*interSet = ks.intersectionSet(ks1);

interSet->showSetBin();

interSet->showSet();

printf("---------------------------------------------\n");

printf("Subtraction \n");

KaderStruct \*subSet = ks.subtractionSet(ks1);

subSet->showSetBin();

subSet->showSet();

printf("---------------------------------------------\n");

delete[] arr;

return 0;

}